경사 하강법

Intro

기본적인 함수 최적화 방법 중 하나인 경사 하강법에 대해 알아보자.

쉽게 말하면 1차 미분계수를 활용해 iterative하게 함수의 최소값을 찾아가는 방법이다.

우리는 이미 gradient에 대한 설명을

에서 했지만, 설명의 연속성을 위해 복습해보도록 하자.

어떠한 다변수 함수 $f(x_1, x_2, \cdots, x_n)$이 있을 때 f의 gradient vector는

$$\nabla f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n})$$

와 같이 정의된다. 즉, gradient는 각 변수로의 일차 편미분 값으로 구성되는 벡터이며, 이 벡터는 f의 값이 가장 가파르게 증가하는 방향을 나타낸다. 이 때 벡터의 크기는 그 증가의 가파른 정도, 즉 기울기를 나타낸다.

예를 들어서 $f(x, y) = x^2 + y^2$의 gradient를 구해보면

$$\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}) = 2x + 2y$$

이므로, 점 $(1, 1)$에서 f의 값이 최대로 증가하는 방향은 $(2, 2)$이고, 그 때의 기울기는 $\|(2, 2)\| = \sqrt{8}$이 된다.

Why Gradient Descent?

경사 하강법은 함수의 최소값을 찾는 문제, 즉 최소값이 최적해가 되는 최적화문제에서 활용된다.

함수의 최소, 최대값을 찾으려면 미분계수가 0인 지점을 찾으면 그만인데, 그러한 방식 대신 경사 하강법을 쓰는 이유는 다음과 같다.

- 실제로 우리에게 주어진 함수가 closed form이 아닐 수 있고 또한 함수가 복잡해 미분계수와 근을 구하기 어려울 수 있다.
- 미분계수 계산보다 경사 하강법이 컴퓨터로 구현하기 더 용이하다고 한다.
- 데이터의 양이 매우 큰 경우 경사 하강법 등의 iterative한 방법이 계산량 측면에서 효율적이라고 한다.

Gradient Descent의 수식

결국 gradient descent는 함수의 기울기를 이용해서 x를 어디로 옮겼을 때 함수가 최소값을 갖는지 알아보는 방식이다.

어떤 점에서 함수의 기울기가 양수라는 것은 x가 커질수록 함수 값이 커진다는 것을 의미하고, 반대로 기울기가 음수면 x값이 커질수록 함수 값은 작아짐을 의미한다.

또한 기울기가 크다는 것은 x의 위치가 최소값이나 최대값으로부터 멀다는 것을 의미한다. (가까울수록 기울기가 점차 완만해져 0으로 수렴할테니까)

따라서 임의의 x에서 기울기가 양수라면 음의 방향으로 이동해야 최소값에 다다를 수 있고, 반대로 기울기가 음수라면 양의 방향으로 이동해야 최소값에 다다를 수 있다. 이를 수식으로 나타내면

$$x_{i+1} = x_i - \text{이동거리} \times \text{기울기의부호}$$

여기서 이동거리는 임의의 x를 업데이트하는 정도라고 보면 된다. 임의의 x에서 gradient의 크기가 크다면 아직 최적해와 멀다는 의미이기에 많이 이동하면 되고, 반대로 gradient의 크기가 적다면 최소값을 지나치치 않기 위해 조금씩 이동해야 한다.

따라서 이동거리는 조정 가능해야 하며 이 때 그 값을 주로 step size라 표현하고 기호로는 λ로 쓰겠다.

따라서 gradient descent의 최종 수식은 다음과 같다.

$$x_{i+1} = x_i - \lambda \nabla f(x_i)$$

이러한 경사 하강법의 문제점은 local minima에 빠질 수 있다는 점이다. 최소값에 다다랐다고 생각했지만 알고보니 옆에 global minimum이 존재하는 경우이다.

또 하나의 문제점은 최적해와 가까워질수록 수렴속도가 느려진다는 점이다. 해에 근접할수록 gradient는 0에 가까워지기 때문에 발생하는 문제이며, 그렇다고 step size를 지나치게 크게 설정하면 global minimum을 찾지 못하고 발산하는 경우가 생길 수 있다.

Gradient Descent의 활용

비선형 연립방정식으로 주어지는 Least square 문제를 경사 하강법으로 해결해보도록 하자.

$$\begin{gathered} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_2(x_1, x_2, \cdots, x_n) = 0 \\ \vdots \\ f_m(x_1, x_2, \cdots, x_n) = 0 \end{gathered}$$

위의 비선형 연립방정식을 동시에 만족하는 $\boldsymbol{x} = x(x_1, \cdots, x_n)$를 구하는 문제는 결국 아래의 식에서 E를 최소화하는 $\boldsymbol{x}$를 구하는 최소 차승 문제로 볼 수 있다. (연립방정식을 LS 또는 SVD로 해결하는 방법은 차후 다룰 예정)

$$E = f_1^2 + f_2^2 + \cdots + f_m^2 = \begin{bmatrix} f_1 & \cdots & f_m \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix} = F^\top F$$

E를 극소로 만드는 $\boldsymbol{x}$에 대한 초기 추정값 $\boldsymbol{x}_0 = (x_{10}, \cdots, x_{n0})$부터 시작해서 아래와 같은 gradient descent 탐색을 반복하면 E의 극소점을 근사적으로 찾을 수 있다.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \lambda_k \nabla E(\boldsymbol{x}_k),\ k \geq 0$$

이 때 ∇E를 직접 구해도 되지만 $E = F^\top F$로부터 $\nabla E = 2J_F^\top F$라는 사실을 이용해 아래처럼 F의 Jacobian인 J_f를 이용해서 해를 탐색할 수도 있다.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - 2\lambda_k J_F^\top(\boldsymbol{x}_k) F(\boldsymbol{x}_k),\ k \geq 0$$

cf) 뉴턴 방법과의 비교

뉴턴법은 방정식 $f = 0$의 해를 점진적으로 찾는 방식이고 경사 하강법은 함수의 극대, 극소를 찾는 방식이라는 점에서 차이가 있다.

그러나 일차 미분을 사용하는 등 그 내부의 원리가 매우 유사하기 때문에 함수 f가 극대, 극소인 점은 $f' = 0$이기도 하므로 뉴턴법으로 $f' = 0$인 점을 구해도 된다.

반면, 경사 하강법은 step size를 조절하는 파라미터은 λ가 필요하지만 뉴턴법의 경우는 그것이 필요 없다는 차이도 존재한다. 뉴턴법에서는 현재의 함수값과 기울기로부터 step size가 자동적으로 결정되기 때문이다. $(\frac{f(x)}{f'(x)})$

또한 뉴턴법은 해 근처에서 수렴속도가 느려지는 문제도 없기 때문에 함수 최적화에서 경사 하강법보다 뉴턴법이 더 효율적이라는 시각이 존재한다.

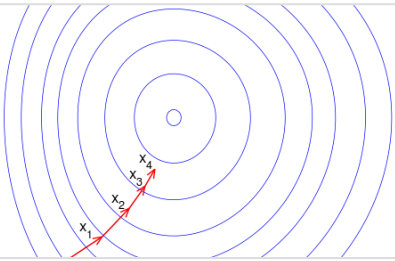
하지만 뉴턴법만으로는 극대와 극소를 구분할 수 없고, 또한 $f'(x) = 0$이라고 해서 반드시 f가 해당 지점에서 극점인 것이 아니기 때문에 Hessian test 등의 추가적인 판단이 필요하다.

▼ 참고

Gradient Descent 탐색 방법

기본적인 함수 최적화(optimization) 방법 중 하나인 gradient descent 방법에 관한 글입니다. Gradient descent 방법은 미분의 개념을 최적화 문제에 적용한 대표적 방법 중 하나로서 함수의

https://darkpgmr.tistory.com/133?category=761008



경사하강법(gradient descent)

Gradient Descent 방법은 1차 미분계수를 이용해 함수의 최소값을 찾아가는 iterative한 방법이다. Step size를 조정해가며 최소값을 찾아가는 과정을 관찰해보자. gradient descent 방법은 steepest descent 방법이라고도 불리는데, 함수 값이 낮아지는 방향으로 독립 변수 값을 변형시켜가면서 최종적으로는 최소 함수 값을 갖

https://angeloyeo.github.io/2020/08/16/gradient_descent.html